

FIG. 1

Prior Art

2/12

18

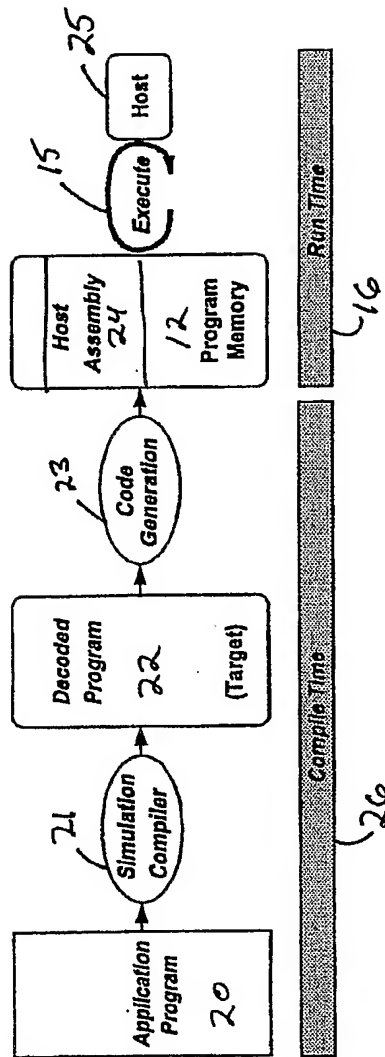


FIG. 2  
Prior Art

3/12

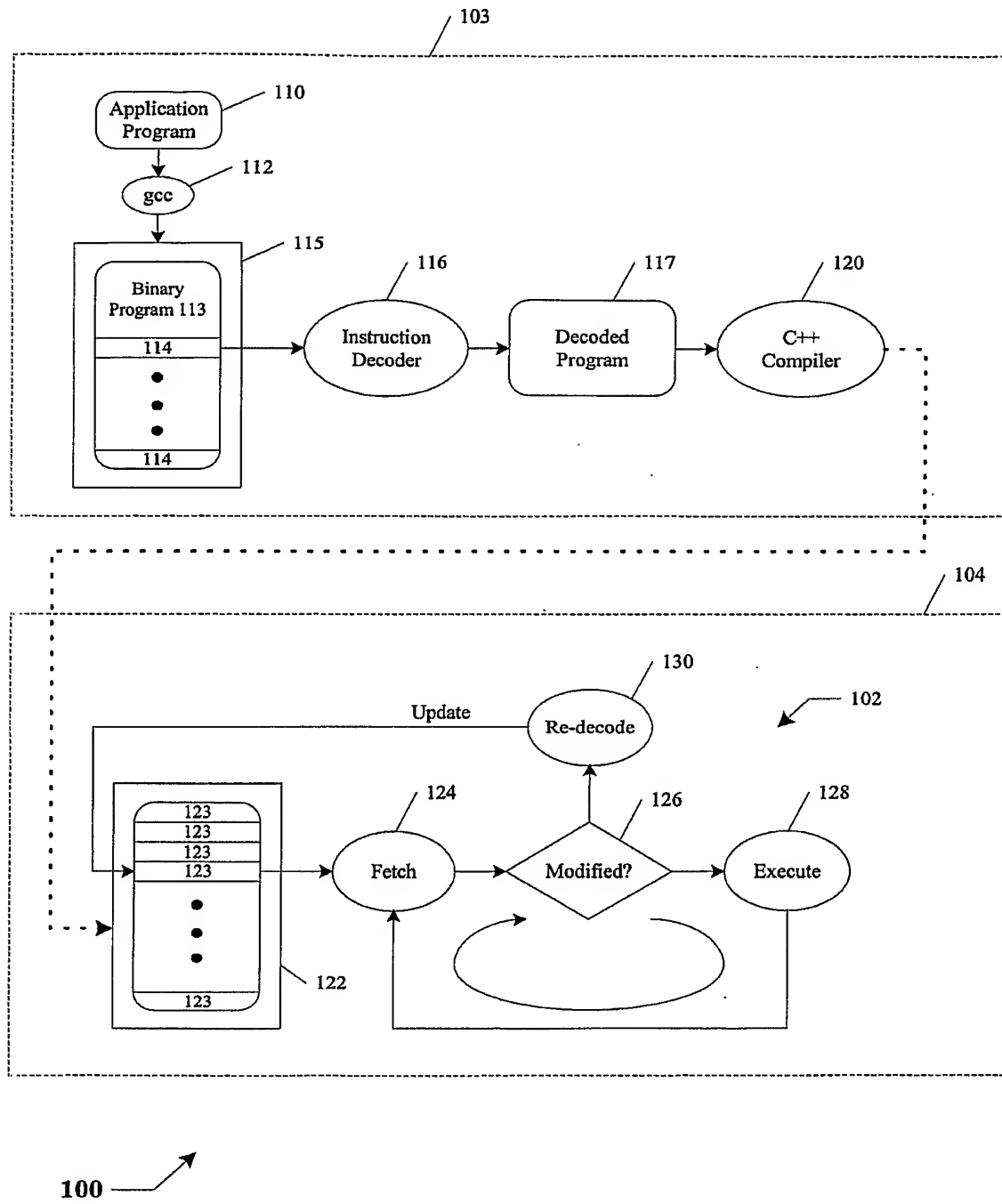


FIG. 3

4/12

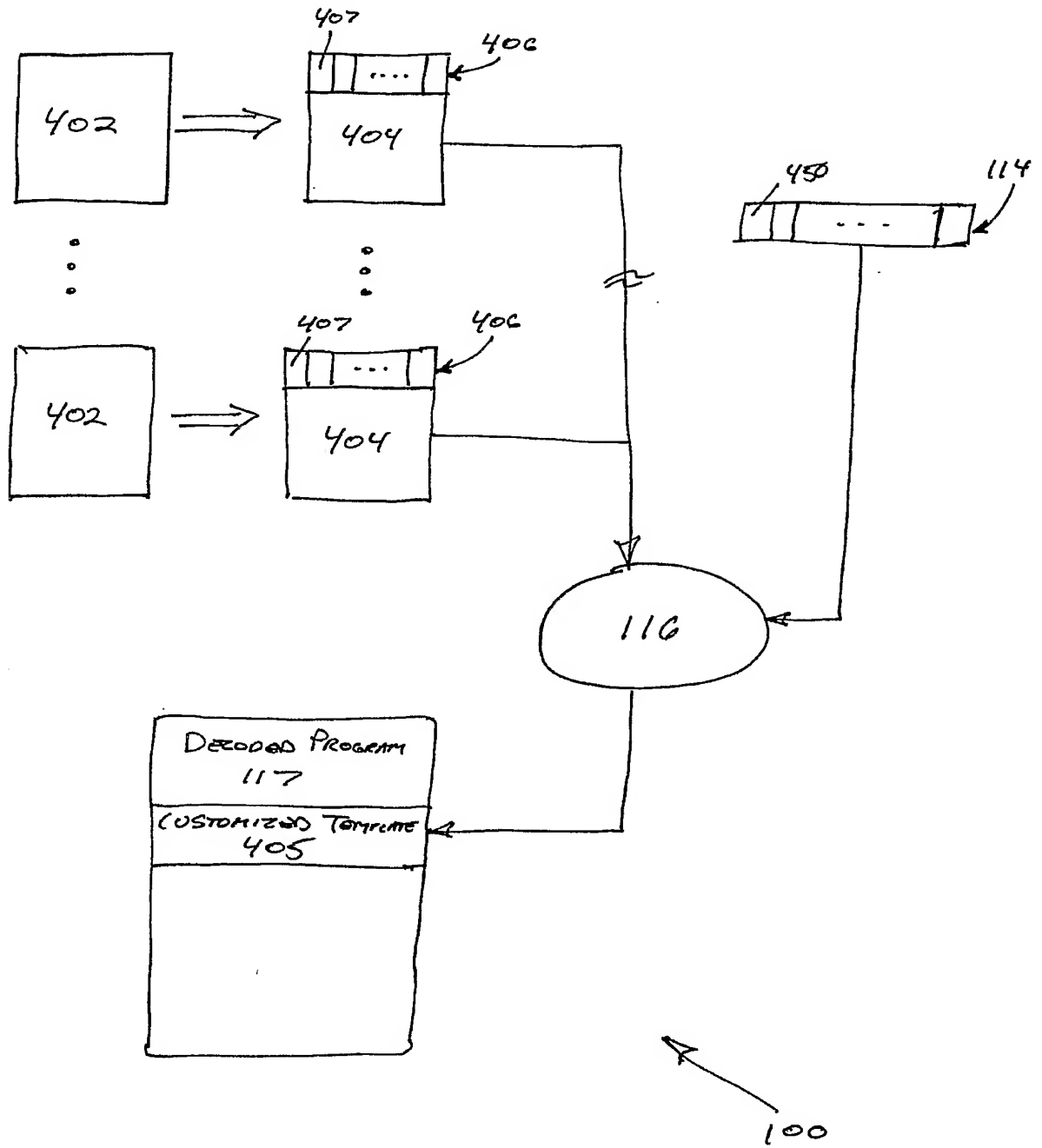


FIG. 4A

5/12

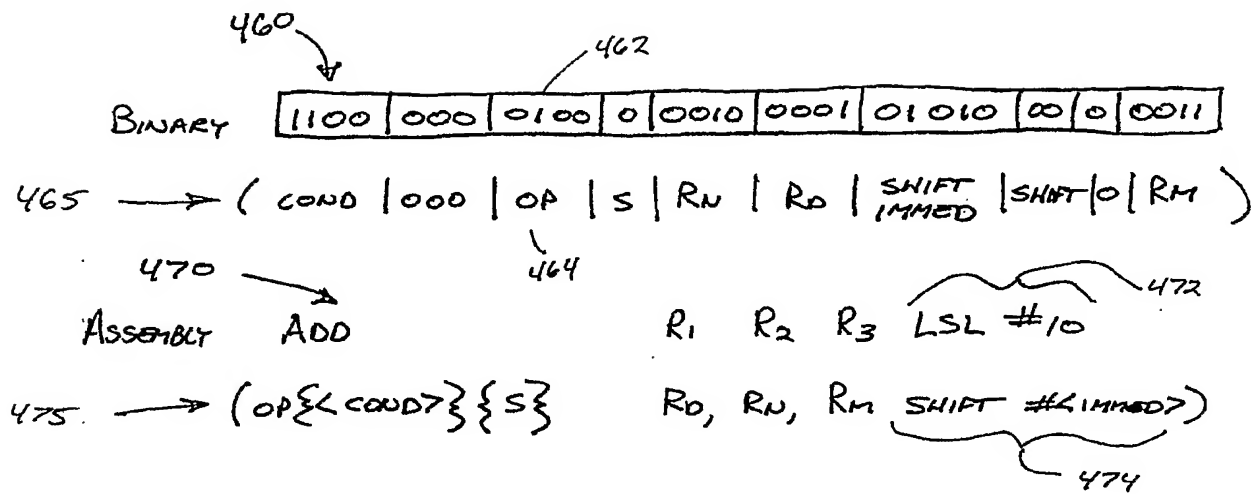


FIG. 4B

6/12

```

#pragma once
#ifndef ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED

#include "SimpresLib/stdIncludes.h"
#include "SimpresLib/instruction/Instruction.hpp"
#include "SimpresLib/val/Reg.hpp"
#include "ARMLib/instruction/lib/Flags.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class Operation, class SBit, class RdIsPC, class ShifterOperandType, class RdType, class RnType>
class DataProcessing : public SimpresLib::instruction::Instruction
{
    ShifterOperandType* _shifterOperand;
    RdType* _rd;
    RnType* _rn;

public:
    DataProcessing(ShifterOperandType* shifterOperand, SimpresLib::val::Reg* rd, SimpresLib::val::Reg* rn)
    {
        _shifterOperand = shifterOperand;
        _rd = (RdType*) rd;
        _rn = (RnType*) rn;
    }

    ~DataProcessing(void)
    {
        delete _shifterOperand;
    }

    virtual void execute()
    {
        if (Condition::exec())
        {
            Word lhs = _rn->getWord();
            Word rhs = _shifterOperand->getValue();
            Word result = Operation::exec(lhs, rhs);
            _rd->setWord(result);
            if (SBit::value() == true)
            {
                //update flags!
                int CF = lib::Flags::getC();
                int VF = lib::Flags::getV();

                Operation::updateC(lhs, rhs, result, &CF);
                Operation::updateV(lhs, rhs, result, &VF);

                lib::Flags::setN(NEG(result));
                lib::Flags::setZ((result)? 0 : 1);
                lib::Flags::setC(CF);
                lib::Flags::setV(VF);
            }
        }
    }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED */

```

FIG. 5A

404

7/12-

```

#pragma once
#ifndef ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED

#include "SimpressLib/stdIncludes.h"
#include "SimpressLib/instruction/Instruction.hpp"
#include "SimpressLib/val/Reg.hpp"
#include "SimpressLib/component/storage/memory/MainMemory.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class LBit, class ElementType, class AddressingMode2_3Type, class RdType>
class LoadStore : public SimpressLib::instruction::Instruction
{
    AddressingMode2_3Type* _addressingMode2_3;
    RdType* _rd;
    SimpressLib::component::storage::memory::MainMemory* _mainMemory;
public:
    LoadStore(AddressingMode2_3Type* addressingMode2_3, SimpressLib::val::Reg* rd)
    {
        _addressingMode2_3 = addressingMode2_3;
        _rd = (RdType*) rd;
        _mainMemory = ARCH_CALL(getMainMemory());
    }

    ~LoadStore(void)
    {
        delete _addressingMode2_3;
    }

    virtual void execute()
    {
        if (Condition::exec())
        {
            Word addr = _addressingMode2_3->getAddress();
            if (LBit::value() == true)
            {
                //Load
                if ((addr % sizeof(ElementType::type)) != 0)
                    FATAL_ERROR("Invalid address for loading!");

                int elemValue = (int) _mainMemory->load<ElementType::type>(addr);
                _rd->setWord(elemValue);
            }
            else
            {
                //Store
                ElementType::type elemValue = (ElementType::type) _rd->getWord();
                _mainMemory->store<ElementType::type>(addr, elemValue);
            }
        }
    }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED */

```

FIG. 5B

404

8/12

```

#pragma once
#ifndef ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED

#include "SimpresLib/stdIncludes.h"
#include "SimpresLib/instruction/Instruction.hpp"
#include "SimpresLib/val/Reg.hpp"
#include "ARMLib/instruction/lib/Flags.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class SBit, class AccumulateBit>
class Multiply : public SimpresLib::instruction::Instruction
{
private:
    SimpresLib::val::Reg* _rd;
    SimpresLib::val::Reg* _rn;
    SimpresLib::val::Reg* _rs;
    SimpresLib::val::Reg* _rm;
public:
    Multiply(SimpresLib::val::Reg* rd, SimpresLib::val::Reg* rn, SimpresLib::val::Reg* rs, SimpresLib::val::Reg* rm)
    {
        _rd = rd;
        _rn = rn;
        _rs = rs;
        _rm = rm;
    }

    ~Multiply(void)
    {
    }

    virtual void execute()
    {
        if (Condition::exec())
        {
            Word result = _rm->getWord() * _rs->getWord();
            if (AccumulateBit::value())
                result += _rn->getWord();

            _rd->setWord(result);
            if (SBit::value() == true)
            {
                lib::Flags::setN(NEG(result));
                lib::Flags::setZ((result)? 0 : 1);
            }
        }
    }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED */

```

FIG. 5C

404

9/12

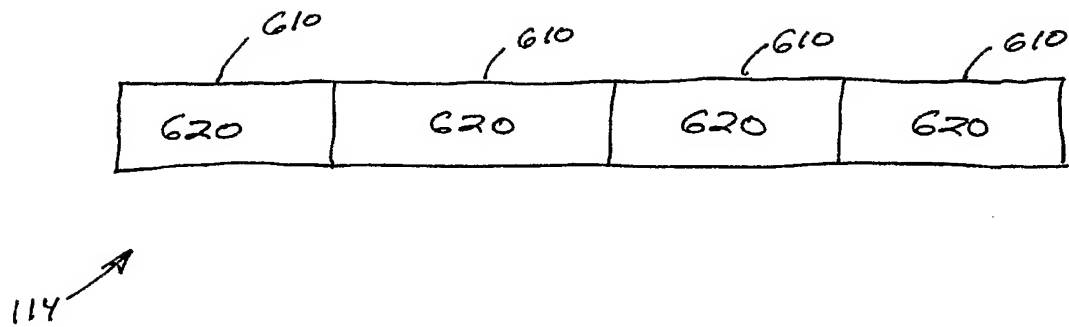


FIG. 6A

10/12

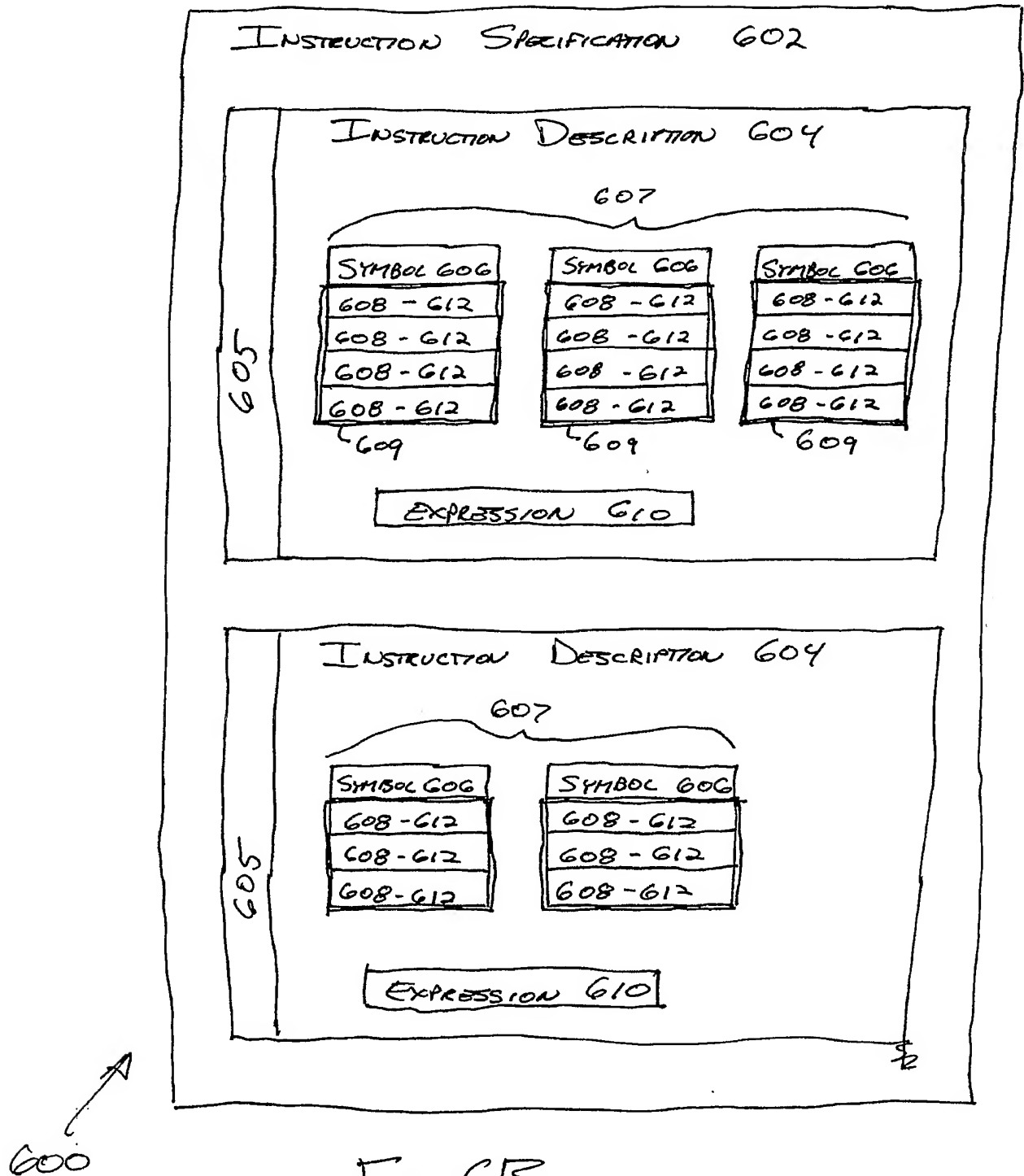


FIG. 6B

11/12

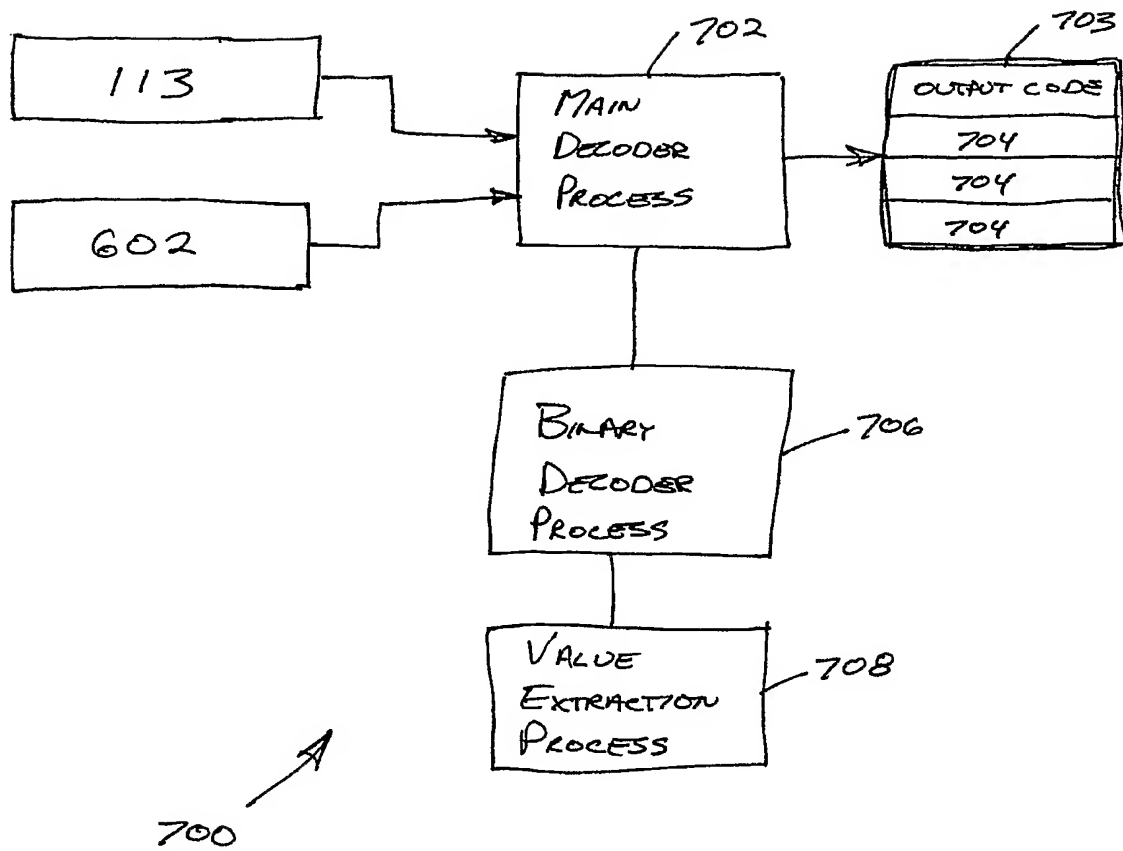


FIG. 7

12/12

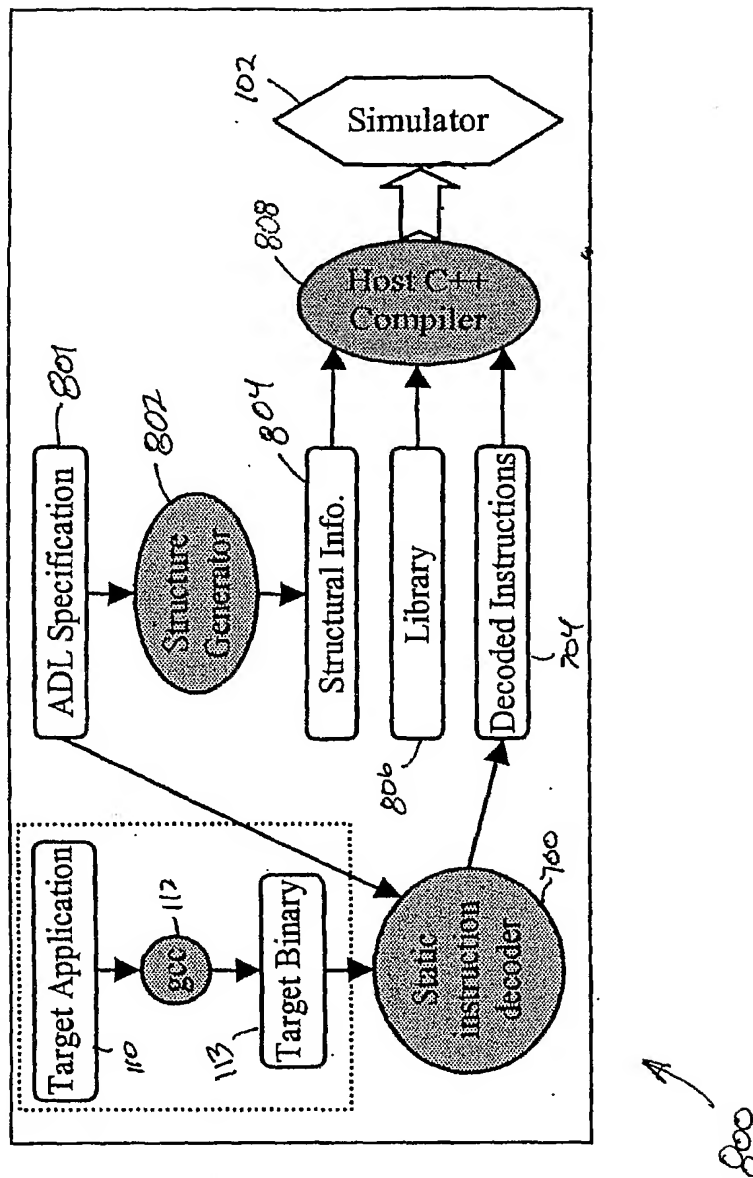


Fig. 8